Faust audio DSP language in the Web

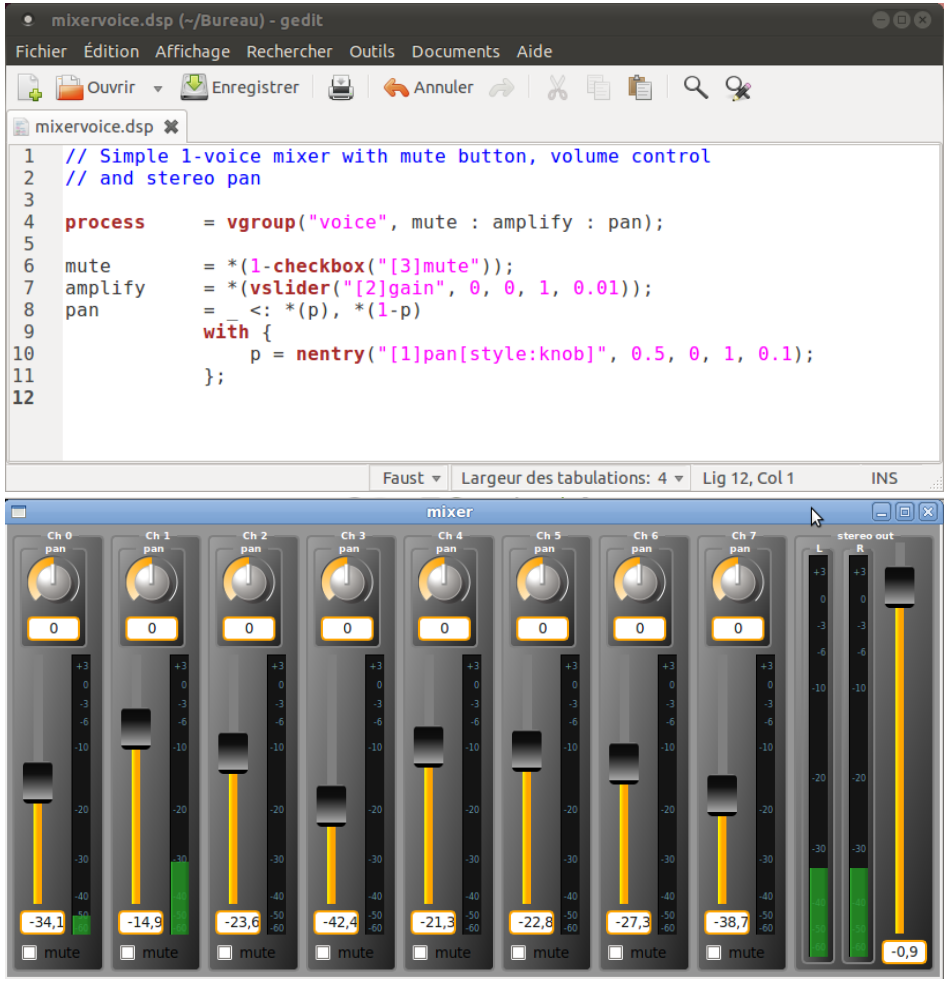
Stephane Letz, Sarah Denoux, Yann Orlarey, Dominique Fober
GRAME
Centre national de création musicale

Linux Audio Conference, Mainz, 2015/04/10

Faust

Speed up audio application and plug-in development

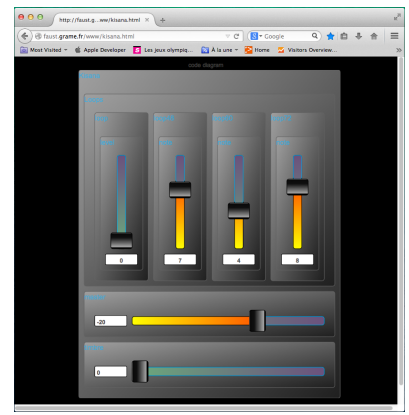
Faust offers an abstract high-level notation to describe DSP algorithms in a concise and effective manner.



Faust

Develop once, deploy everywhere

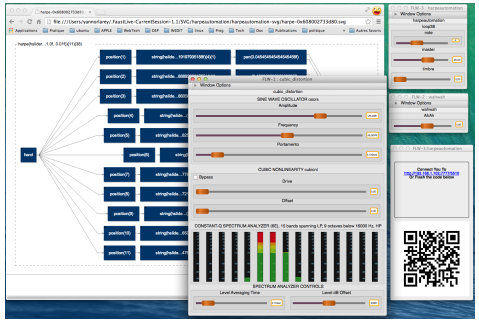
Audio applications designers have to deploy their work on a variety of platforms (Linux, OSX, Windows, Android, iOS, embedded devices, etc). One of Faust strong idea is to write the DSP once en easily deploy it on a wide number of systems.



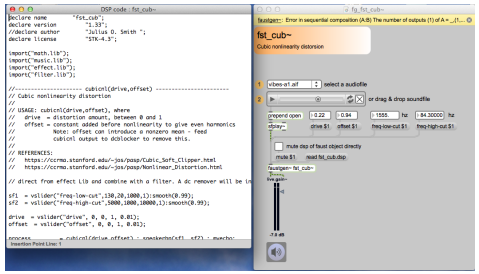
Faust

Make Faust compilation technology widely usable

Faust compilation technology is accessible using the **online compiler**, the **embedded compiler library version**, or the **FaustWeb remote compilation API** that produces various target binaries.



FaustLive with FaustWeb access



Faust in Max/MSP (faustgen using libfaust.so)

Online compiler

Audio on the WEB

WEB technologies like **asm.js**, **Web Audio API** or **Web components** aim to change the way we design, publish and share musical applications. Using this technologies *procedural content* can now be shared and combined as easily as *multimedia content* ! Grame offers several Web technologies :

- **libfaust.js + asm.js target (emcripten + Faust backend)** : embeddable JavaScript/asm.js Faust compiler
- **FaustWeb** : remote multi-targets compilation API
- **Faust Playground** : simplifying Faust programs design

Audio on the WEB

Targeting the Web Audio API (1)

The **Web Audio API** is a high-level JavaScript API for processing and generating audio in Web applications :

- native optimized C++/assembly nodes
- JavaScript/asm.js **ScriptProcessor** nodes
- connected to create an audio generating/processing graph

Audio on the WEB

Targeting the Web Audio API (2)

How to generate **ScriptProcessor** nodes ?

- they can be "manually written" in pure JavaScript
- or in asm.js for better performances (but this is difficult...)
- or automatically generated from DSP code already written in C/C++... (emscripten)
- or **automatically generated from a Domain Specific Language**

Audio on the WEB

Asm.js code generation (1)

Asm.js is developed by Mozilla along with **Emscripten** :

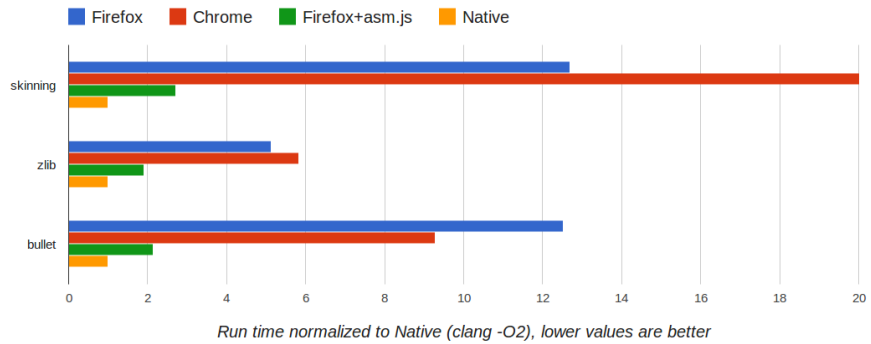
- **asm.js** : an extremely restricted subset of JavaScript that provides only strictly-typed integers, floats, arithmetic, function calls, and heap accesses (using typed arrays).
- **asm.js** variables, computation, return values types are annotated
- **asm.js** can easily be optimized
- future extensions like **SIMD.js** (vectorized types in JavaScript)

Audio on the WEB

Asm.js code generation (2)

```
function GeometricMean(stdlib, foreign, buffer) {
  "use asm";

  var exp = stdlib.Math.exp;
  var log = stdlib.Math.log;
  var values = new stdlib.Float64Array(buffer);

  function logSum(start, end) {
    start = start|0;
    end = end|0;

    var sum = 0.0, p = 0, q = 0;

    // asm.js forces byte addressing of the heap by requiring shifting by 3
    for (p = start << 3, q = end << 3; (p|0) < (q|0); p = (p + 8)|0) {
      sum = sum + +log(values[p>>3]);
    }

    return +sum;
  }

  function geometricMean(start, end) {
    start = start|0;
    end = end|0;

    return +exp(+logSum(start, end) / +((end - start)|0));
  }

  return { geometricMean: geometricMean };
}
```

Exemple of asm.js module



asm.js benchmark (2 to 3 times slower than native code...)

Audio on the WEB

Asm.js code generation (3)

Generating asm.js with Emcripten :

- Emcripten C/C++ to JavaScript (asm.js) compiler developed by Mozilla starting in 2011
- Facilitates the port of huge C/C++ codebase on the Web

Audio on the WEB

Asm.js code generation (4)

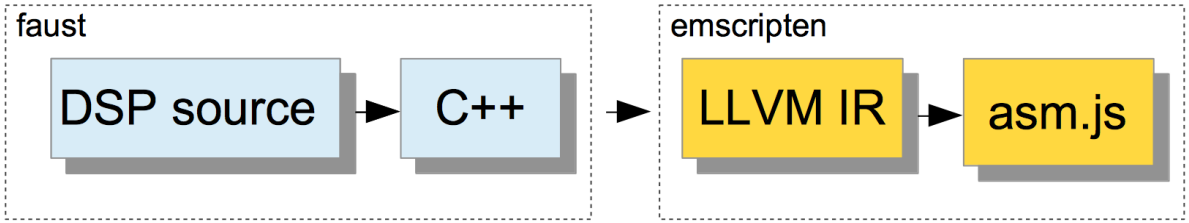
Asm.js backend in Faust compiler : produces the asm.js module + some pure JavaScript helper functions

```
......
function getValue(dsp, offset) {
    dsp = dsp | 0;
    offset = offset | 0;
    return +HEAPF32[dsp + offset >> 2];
}

function compute(dsp, count, inputs, outputs) {
    dsp = dsp | 0;
    count = count | 0;
    inputs = inputs | 0;
    outputs = outputs | 0;
    var output0 = 0;
    var fSlow0 = 0.;
    var i = 0;
    output0 = (HEAP32[outputs + (0 << 2) >> 2] | 0);
    fSlow0 = +(+(4.65661e-10 * +(+(HEAPF32[dsp + 8 >> 2]))));
    for (i = 0; (((i | 0) < (count | 0)) | 0); i = (((i | 0) + 1) | 0)) {
        HEAP32[dsp + 0 + (0 << 2) >> 2]
            = ~~(((12345 + ~~((imul(1103515245, (HEAP32[dsp + 0 + (1 << 2) >> 2] | 0)) | 0))) | 0));
        HEAPF32[output0 + ((i | 0) << 2) >> 2]
            = +(+(+(+(fSlow0) * +((HEAP32[dsp + 0 + (0 << 2) >> 2] | 0)))));
        HEAP32[dsp + 0 + (1 << 2) >> 2] = (HEAP32[dsp + 0 + (0 << 2) >> 2] | 0);
    }
}
......
```

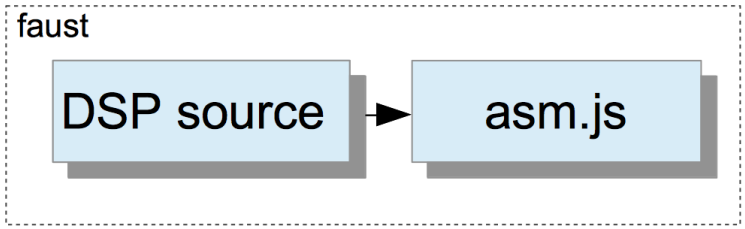
Audio on the WEB

Asm.js code generation (5)

Static compilation chain (Faust DSP to asm.js) allows to generate self-contained HTML pages.



- using emscripten as an intermediate step :



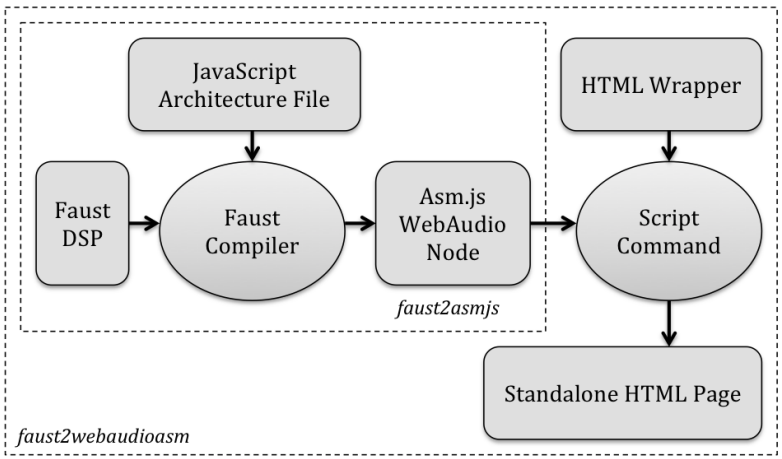
- or using direct asm.js code generation

Audio on the WEB

JavaScript compilation : asm.js generation

Static compilation chain scripts :

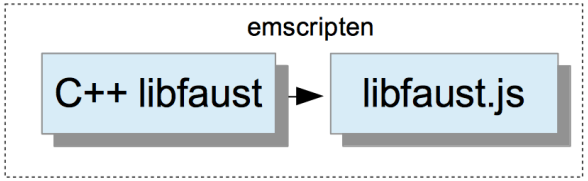
- takes Faust DSP, compile it to asm.js, wraps it with additional JavaScript code to obtain a fully functional Web Audio node.
- wraps the Web Audio node in a HTML template to obtain a self-contained DSP node in the page



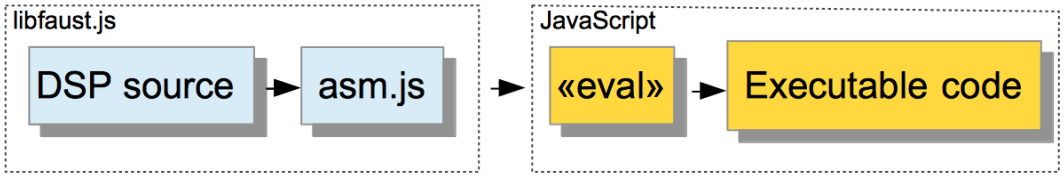
Audio on the WEB

JavaScript compilation : asm.js generation

Dynamic compilation chain (libfaust.js + asm.js backend) allows to embed the complete compilation chain in the browser :



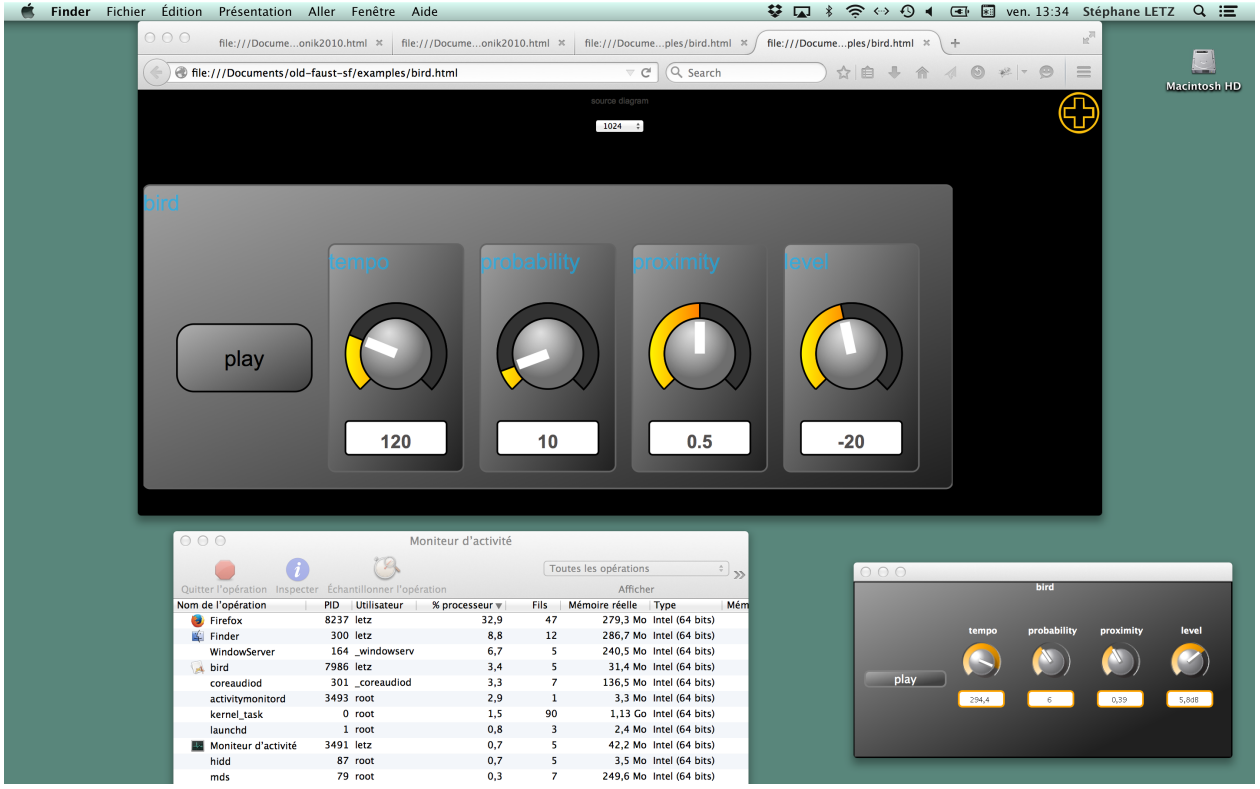
- first compile C++ **libfaust** for the Web (**libfaust.js**)

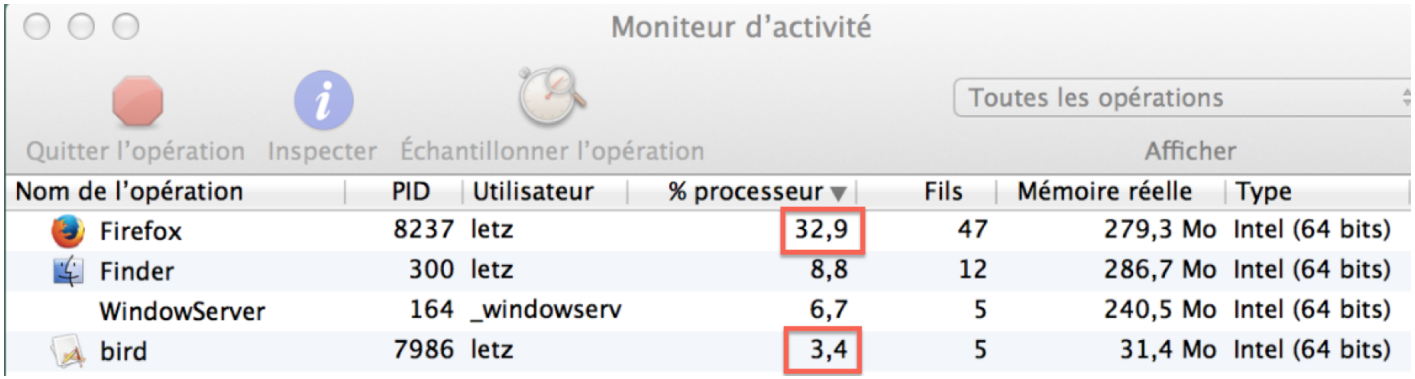


- compilation of an asm.js module happens at parse time of the source code. If parse time is triggered with ‘eval‘ then **dynamic compilation occurs**.

Benchmark of a CPU light application

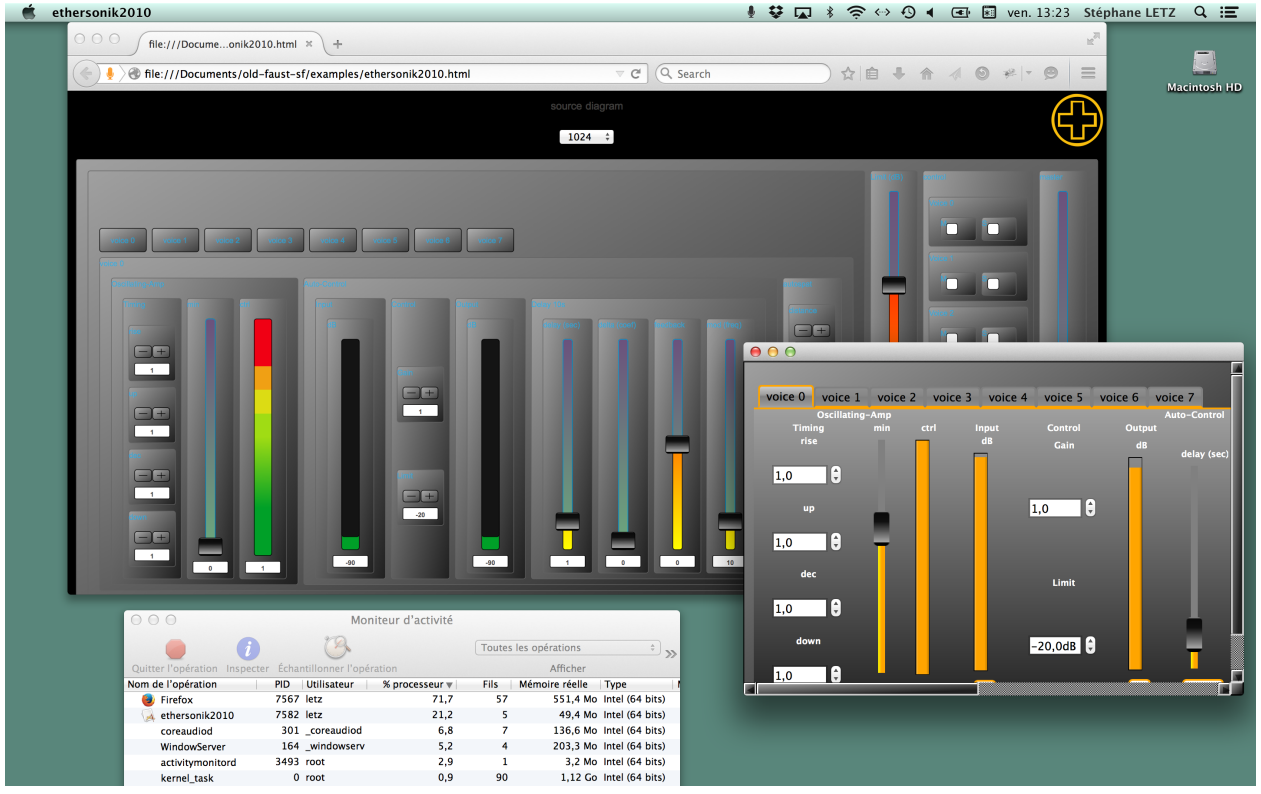
Bird ported on the Web

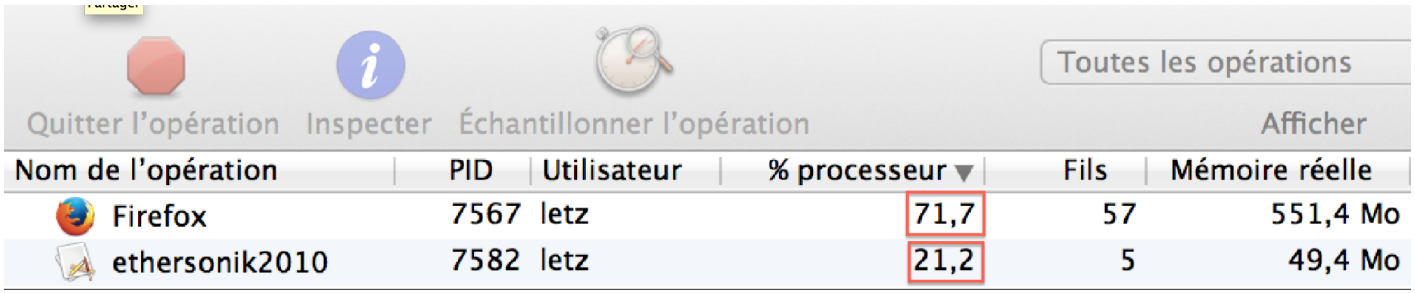




Benchmark of a CPU heavy application

Yann Orlarey Ethersonik ported on the Web





Demo

- faust2webaudioasm script
  - From harpsichord.dsp to harpsichord.html

- faust2asmjs
  - Harpsichord
    Thomas Cipierre & Laurent Pottier (Saint-Etienne, France)
  - foo-yc20
    Sampo Savolainen (Helsinki, Finland)

- libfaust.js
  - FaustPlayground : create Faust patches online

Conclusions and perpectives

- still some issues with the Web Audio API : implementation, performances CPU/latency (audio workers : moving the ScriptProcessor nodes in the audio thread)
- really usable for serious work? still to be proved...
- but at least already usable for deployment, distribution, teaching purposes...

GRAME – R&D

Softwares

Softwares developed in different research projects are freely available under GPL/LGPL licenses :

- Faust : `http://faust.grame.fr` :
  - Faust : `git.code.sf.net/p/faudiostream/code`
  - FaustLive : `git.code.sf.net/p/faudiostream/faustlive`
  - FaustWorks : `git.code.sf.net/p/faudiostream/faustworks`
  - FaustWeb : `git://git.code.sf.net/p/faudiostream/faustweb`

GRAME – R&D

Recent publications on Faust

- Denoux, Letz, Orlarey, Fober 2014 : *FAUSTLIVE: Just-In-Time Faust Compiler... and much more*. LAC 2014.
- Denoux, Letz, Orlarey, Fober 2014 : *FaustLive un compilateur à la volée pour Faust ... et bien plus encore*, Journées d'Informatique Musicale, Bourges.
- Brune de Chiffreville 2014 : *Using Faust with Ros*. Rapport de Stage, GRAME.
- Denoux, Letz, Orlarey, Fober 2015 : *Composing a web of audio applications*, WAC 2015, Paris.
- Letz, Denoux, Orlarey, Fober 2014 : *Faust audio DSP language in the Web*, LAC 2015